
What AI speeds up, and what it can't replace

Joshua Campbell · Atlas

I'm Josh, and I run Atlas in Christchurch. Atlas began in video production, and I brought around a decade of software experience to the business. This guide explores what happens when that experience meets AI: where the work gets faster, and what still needs human judgement, trust and responsibility.

When I say "what it can't replace", I mean in our practical use today. That does not mean it will never replace those things. The focus here is implementation: how a tool changes the work, what it improves for a business and its customers, what new work it creates, and who takes responsibility when something goes wrong.

The experience behind these examples

I have worked with enterprise brands across Australia, New Zealand, the UK and the US. My involvement has ranged from contributing within teams and designing solutions to taking full responsibility for project delivery. These organisations represent experience across my career; they are not all current direct Atlas clients.

Those brands include Maersk, Vodafone, Telstra, QUBE, OZ Minerals, One.nz, University of Adelaide, Discovery Parks Australia, City of Canberra, City of Melbourne, City of Adelaide, City of Alice Springs, Multiquip, MTData, AusHealth, SolarReviews, Barossa Fine Foods, Ferrari Formalwear, Navair, Edgewater Hotel and Exceltech / Excel-IT.

How to read this guide

We begin with direction and the reality of business systems, then look at Atlas as a practical example. The middle sections explain how AI speeds up learning, finding information and building software. The final sections examine engineering responsibility, buying decisions, reputation, pricing and human connection.

The stories and observations are drawn from my work. Examples that illustrate a technical principle are explained as examples, rather than presented as proof of a particular product's performance.

Direction before speed

The person using the tools

Why Atlas? We map and navigate. A clock tells you how fast time is passing. A compass helps you decide whether you are heading somewhere worthwhile. AI can help us explore options and do the work, but choosing what matters to us remains our responsibility.

The fastest way to get where you want to go is to understand where you are going. Putting everything you have into moving flat out is a poor outcome if you arrive somewhere you never wanted to be.

Consider a beautiful report nobody reads. Producing it faster does not make it useful. A report is an output; a better decision is an outcome. Before automating a report, establish which decision it is meant to support. The same distinction applies to software, content and how we spend our days.

Curiosity is enough of a reason

Whatever your age, you are allowed to learn something without turning it into your next business. You are allowed to change direction. A difficult year does not mean you missed your only chance.

AI is a major change, but it will not be the only change in our lives. Exploring it can be fun and terrifying. Curiosity is a good reason to begin. Feeling that you must keep up with everyone on the internet is a miserable reason. Learning a language, understanding something your child cares about, or finally making sense of a technical subject are worthwhile outcomes too.

Activity, productivity and rest

Generating more content or code can look impressive without improving anything. Imagine producing 50 social posts before deciding who they are for or what they should achieve. You now have 50 posts to check. Compare that with answering one real customer question and checking whether the answer helped.

Rest is not laziness. You are still making the decisions. If you are exhausted, increasing the flow of decisions coming towards you may be the last thing you need. You are the captain; the point of a faster ship is not to fall asleep at the helm.

Have you used AI to produce something quickly, then wondered whether you needed it? Which task in your week creates activity without changing an outcome? What would you explore if nobody could see the result and you did not have to monetise it?

The reality inside businesses

New tools arrive; existing systems remain

In my work with enterprise brands, the gap between the pace of technology and the pace at which a business can change is striking. Many businesses pay substantial amounts for off-the-shelf software that covers about 80% of their problems. That figure describes what I have encountered; it is not an industry benchmark.

The remaining work often lives in spreadsheets, paper systems and, bless their souls, Microsoft Outlook. Those tools are familiar and flexible. They often handle the tasks the main system does not cover.

Think of the layers that accumulate: manual records, packaged software, connected systems and now AI assistance. This is a conceptual progression, not a dated history or a claim that every business follows the same path. The layers coexist. New technology arrives while earlier systems, habits and obligations remain.

Internal friction becomes customer friction

For a business, the gaps can mean entering the same information twice, chasing approvals or paying staff to bridge systems manually. For customers, that can mean repeating their details, waiting for answers or receiving inconsistent information.

An internal workaround does not necessarily stay internal. It can shape the entire customer experience. A staff member may be doing their best while the systems around them make a simple request unnecessarily difficult.

Adding an AI feature on top does not automatically fix the problem. First understand where the work gets stuck. An integration, a simpler process or a clearer handover may help more than another generated answer. AI needs to earn its place through the improvement it makes.

Keep the parts that work and improve the part that creates friction. The useful question is whether the change removes work and improves the experience, rather than whether the system now has an AI feature.

Where does your team bridge the gaps between systems? Does a customer have to repeat information because your tools do not share it? What specific handover would you improve first?

How Atlas grew from a real problem

Software and creative work together

Atlas started as a small video production business in Christchurch. As it grew, creating content was only part of the challenge. We also had to manage it, distribute it and keep track of what was going where.

With around ten years of software experience, I began building tools for the work we were already doing. AI accelerated that process. Over time, the internal tools grew into a broader platform.

The starting point matters. We had people doing real work and a problem we could see. We could ask whether a change made their day easier. That provides a better test than whether a new feature looks impressive.

Consider a content workflow: material arrives, somebody checks it, somebody decides where it belongs, and somebody publishes it. Those steps involve permissions, stored media, status and connections to other platforms. AI might help draft or classify material within that flow. The software still has to track the work and enforce who may do what. This is a workflow example, rather than a promise about a particular feature.

What we do today

Atlas combines a platform of connected business applications with bespoke websites, business software and mobile apps. Integration and automation connect existing systems and reduce repeated work. We also continue the creative work: video, content and digital experiences.

The technical and creative sides affect the same person. A technically correct system can still be confusing. A beautiful interface can still fail to solve the problem.

A customer might ask for a website, while the actual work involves bookings, customer records, notifications and how staff manage the booking afterwards. The visible website is one part of that wider system. Understanding the work comes first.

Our competitive advantage is that we love it and we're having fun every day.

That is a personal commitment, not a claim to secret technology. The tools are becoming available to more people. We keep pulling the thread because we enjoy understanding how things work and making them better. Software experience helps us decide what to ask for, how the pieces should fit together and whether the result is dependable.

How Atlas fits together

Shared foundations for a small team

Atlas uses shared capabilities across applications, so each app does not have to invent its own account system and permissions model. The following is a conceptual explanation of responsibilities, rather than a deployment diagram or security audit.

Applications

One supports scheduling and social content. Horizon handles website content. Astra handles forms. Commerce supports selling products. These are examples from the platform, rather than its complete application catalogue.

Aether: the shared core

Aether manages tenants, users, roles and permissions, alongside subscriptions and plans. A tenant is one business's space within the platform. Shared identity and access rules provide a common foundation for the applications people use.

Connections and workflows

APIs are interfaces through which software talks to other software. Vector handles workflows. The connection needed depends on the application and the job it must do.

For example, a person submits information through a form. The application needs to know which business the information belongs to and who may access it. A workflow might notify somebody or pass approved information to another system. The exact workflow varies, but identity and permissions must remain deliberate throughout.

Reuse also creates responsibility

Shared foundations help a small team avoid repeating the same work in every application. They also mean changes to that foundation deserve care: several applications may depend on the same behaviour.

AI can accelerate implementation within this architecture. Somebody still has to choose the boundaries, check the behaviour and take responsibility for releasing a change. The benefit of reuse depends on those shared parts being dependable.

If several tools in your business share the same customer records or login system, what would happen if that shared component changed? Which workflows would need checking before the change reached users?

Learning becomes a conversation

What AI speeds up

One useful way to think about AI is as a translator between a subject and the way you learn. You can ask for plain language, a worked example, a visual explanation or a different angle when the first explanation does not land.

You do not need to get everything from one perfect prompt. Pull the thread of curiosity. Explain which part you do not understand, ask follow-up questions and ask the tool to test your reasoning. Then check important claims against reliable sources. Fluency and correctness are different things.

A worked example: understanding an API

An API is an agreed way for one piece of software to ask another piece of software for information or an action.

For example, a website can ask a scheduling system for available booking times. It sends a request in a format the scheduling system understands and receives a response. That gives you a starting point for understanding how the two systems work together.

Now ask: what happens if the scheduling system is unavailable? That question moves the learning beyond the initial explanation. You can explore errors, retries and what the customer sees when a booking cannot be completed.

A learning conversation can develop as your understanding grows. You might begin with “Explain it simply and give me an example I recognise.” Then try “Ask me a question and check my reasoning.” Finally, ask “What am I missing, and where can I verify it?”

Easier entry still needs practice

AI can make an unfamiliar discipline easier to enter. It does not remove the need for practice, sound sources or people who can challenge your understanding. The goal is to become able to explain and use the idea yourself, rather than simply recognise an answer that sounds convincing.

Choose a subject you have been avoiding because it feels too technical. Ask for a familiar example, explain the idea back in your own words, and identify one claim you can check independently. What follow-up question would take you beyond the first explanation?

The email I couldn't find

Finding the original information

I needed to find an email that had been used as a quote. Not finding it could have cost us \$8,000. I spent about half an hour searching. I knew what I was looking for, but I could not get back to it.

Then I asked my agent. It found the email in about 20 seconds.

That is an unglamorous use of AI: getting the right piece of information back into a human decision. In that moment, it was enormously useful.

Why the original mattered

Knowing that a message exists is different from remembering the exact words its sender used. That gap can make a search frustrating even when you understand the issue well.

In this case, I needed the email itself, its wording and its context. A generated summary that sounded right would not have been enough. The benefit was recovering the original record so I could inspect it and use it in the decision.

That is a useful habit for any AI-assisted search: open the source and check that it answers the actual question. An answer can sound relevant while missing a qualification, a date or the surrounding context that changes its meaning.

Measure the problem resolved

The half-hour search and the 20-second result are my timings from one incident. They are not a benchmark or a promise that every search will take 20 seconds. The value came from resolving a specific problem.

The same principle carries into software development. Producing something quickly is useful only if the result holds up when you inspect it. Speed should help you reach dependable information or a dependable outcome sooner.

Think of a message or document you know exists but struggle to retrieve. How would you describe its purpose and context? Once an agent found a candidate, what would you check before relying on it?

Faster code still needs judgement

Delegating implementation without losing responsibility

You may have heard that developers no longer write code. I would not make that claim about everyone. What is changing is the balance of the work: more implementation can be delegated, while more engineering attention moves to specification, review and verification.

An agent needs context. Which patterns does the project use? What is it allowed to change? How will we know it is finished? What should happen when a test fails? These remain recognisable engineering questions.

AI can help produce draft implementations, tests, documentation, alternative approaches and explanations of unfamiliar code. The engineer still owns the problem and its constraints, architecture and permissions, verification and release, and what happens when the system fails.

A report-export button

Suppose a customer asks for a button that exports a report. Generating a button and a database query may be quick. But the request leaves important questions unanswered:

- Who may press it, and which company's data should appear?
- What if the report includes private information?
- What happens when there are a million rows?
- Does a failed export explain what happened to the user?
- Can the numbers be reproduced and checked?

A screenshot can show that a button exists. It cannot prove that these boundaries work. A plausible interface is only one part of the implementation.

Leading the work

Give the agent a bounded task, provide the project conventions, inspect the change, run meaningful checks and decide whether the result is ready to release. This is how delegated implementation becomes work somebody can stand behind.

Parallel agents can increase the amount of work in progress. They also increase the coordination required. Counting agents is not the same as counting independent engineers.

AI can help build the idea you picture. Keeping it reliable for people who depend on it remains engineering work.

If you were reviewing the report-export feature, what evidence would you need beyond a screenshot? Which failure would matter most to the people using it?

Background reading from the presentation notes: Anthropic, "Effective harnesses for long-running agents" (26 November 2025), on incremental work, continuity and verification. The report-button example is my engineering explanation.

[Read the background article](#)

One platform. Separate business data.

Where engineering judgement becomes concrete

“Multi-tenant” describes a platform used by several organisations while each has its own space. Two businesses can share an application without sharing each other’s private information.

In Atlas, Aether provides the core for tenants, identity and permissions. Applications build on that shared foundation. This explanation describes the responsibility of those parts; it is not an independent security audit of Atlas.

A request from Business A

Imagine Business A asks for a report. The system must establish who is making the request, which business they are acting for, and whether they are allowed to export that information.

Hiding Business B in a dropdown does not provide the protection. The server must enforce the boundary. Otherwise, a different request could bypass what the interface appears to allow.

Now imagine a developer changes the report query. The report still works for the person testing it, but the query no longer restricts the results correctly. The page may look completely normal while the underlying behaviour is wrong.

A meaningful test attempts to request another business’s data and confirms that the system refuses. It tests the boundary that matters, rather than only confirming that the ordinary path produces a report.

What this means for a small team

Shared foundations reduce repeated work. AI can lower the effort required to implement new features. Together, they can help a small team build more.

That does not mean every small team can build any system, or that the responsibility disappears. The implementation barrier is lower, while deliberate technical design and verification still protect the people using the result.

This is where the distinction between activity and productivity becomes tangible. Producing more features is useful when they solve the right problem and respect the constraints that make the system safe to use.

For a system you use or manage, who should be allowed to see each kind of information? How would you check that those rules hold even when someone makes an unexpected request?

Better products still have to earn trust

Buying, marketing and reputation

Would you rather be sold something, or have a problem solved? A convincing demonstration and a recognisable customer logo can make a product feel safe. They do not establish whether it solves your problem, handles your data appropriately or lets you leave when you need to.

Imagine an AI tool promises to reduce administration. Before buying, identify the task it improves, how much checking remains, what happens when it is wrong and whether you can export your information. For the business, does it remove work or move it elsewhere? For the customer, does the experience become easier, clearer and more trustworthy?

Builders should ask the same questions. Online reach creates opportunities far beyond your postcode, but it does not guarantee demand or profit. Understanding a problem and earning trust still matter.

The two-day thought experiment

Suppose I could build a software-as-a-service product in two days with better features, capabilities, actions and integrations than anything else on the market. Even then, I would not have recreated the established brand's trust.

I would still need to reach its customers, tell the story and give them a reason to believe me. Building software does not automatically make people aware of it, comfortable with it or willing to switch. This is a thought experiment about adoption, not a measured claim that every mature product can be surpassed in two days.

Customers ask whether they know the business, whether someone will help when the product breaks, whether switching is worth the disruption, and whether they can trust it with something important.

Marketing helps people discover the product. Storytelling explains why it matters to their situation. Reputation provides evidence that the promise holds up over time. A technically better product can still be commercially invisible.

Trust after the demo

A polished demo earns attention. Afterwards, people need reliable software, a clear understanding of what happens to their information, and support when something goes wrong. Keeping promises, resolving problems and taking responsibility are how reputation grows. Engineering and the customer relationship belong in the same conversation.

What would persuade you to switch from an established product to a new one? Which promises would you need to see demonstrated over time?

When AI becomes the price benchmark

A price needs context

A CFO from a company with nine-figure annual revenue sent me a screenshot of a Google AI Overview about New Zealand software developer contractor rates. He questioned why a project worked out to \$340 per hour when broken down that way.

I increasingly find myself explaining Atlas's value because a quoted price is put into tools such as ChatGPT or Gemini. The resulting answer becomes an anchor in a real commercial conversation. This particular screenshot came from Google's AI Overview; it was material sent to me, rather than a verified rate card.

Even the wording needed scrutiny. The principal/specialist line included "\$125-\$1,500+ per day" and then "\$850+ per day". A polished answer still needs checking before it becomes the basis for a decision.

A contractor benchmark does not price a project

To compare prices meaningfully, we need the scope, responsibilities, deliverables and commercial terms. Software projects can include discovery, architecture, integration, testing, deployment and ownership of the result. A generic hourly contractor figure alone does not describe that package.

AI advice can influence negotiations and what people get paid. The judgement of the person using the advice is consequential.

The \$30,000 website example

Another Christchurch client paid a reputable company \$30,000 for a brochure website design, a logo and a one-page digital strategy. Delivery took two weeks. Assuming five days a week and eight hours a day gives 80 hours. Dividing \$30,000 by 80 produces \$375 per assumed hour.

The arithmetic is correct; the labour assumption is unverified. I do not know the supplier's actual hours, team size or internal costs. Two weeks of elapsed delivery time is not a timesheet, and \$375 is not its established hourly rate.

A guessed hourly number leaves out scope, quality, responsibility, rights and support. Hourly analysis can help when actual effort and terms are known. Elapsed time alone does not establish a project's value.

Reputation, familiarity and whether people like and trust you influence the conversation too. We still have to communicate the value of the work. An AI answer cannot infer that relationship from a price alone. A quote includes responsibility for an outcome, not simply time spent generating code.

When comparing a quote with an AI-generated benchmark, are you comparing equivalent work and terms? Which assumptions about effort, ownership or support have you not checked?

The reason we show up is human

What would you make time for?

AI can help organise an event, write a speech or make music. But the reason I value being in a room with people is the experience of someone taking a risk, telling a story and responding to those in front of them.

An orchestra is more than the notes on the page. A conversation is more than a transcript. A relationship is more than an organised contact record. Those are experiences I want technology to support.

Reading a guide gives you access to ideas. Being with others can also give you the chance to ask an unexpected question, respond to a person's story and form a relationship. Consider what you value in shared experiences that a written summary does not provide.

Lower barriers, worthwhile opportunities

I am optimistic about tools that make learning and experimentation easier to begin. Imagine a biology student getting help to understand unfamiliar analysis code. They may be able to explore an idea sooner. They still need sound methods, validation and the relevant scientific expertise.

The exciting part is making the first step more accessible. The value of speed is what it makes possible for people: curiosity, useful work and time for experiences they care about.

A starting point for your own work

Choose one small problem you could explore this week, something you genuinely care about. Ask a question and follow up on the part you do not understand. Decide what a useful result would look like before asking for more output. Then check what you receive.

AI can help you understand something unfamiliar, find the record you need and build an idea faster. I have seen those benefits in my own work. Responsibility for what matters to you and for the people affected by what you build remains with you.

If the time you save becomes a walk, a conversation or an evening when you switch off, that counts too. You do not have to win an AI race. You get to decide where you are going.

What is one problem worth your attention? How would you recognise a useful result? If solving it gave you time back, what would you choose to do with that time?